

```

//!- cogging.rad -----
ABSTRACT = a cogging;
CONCRETE = cogging;
VERSION = $Revision: 1.1 $;
DESC = This ADO talks to the RF Synchro Synthesizer board;
ARGS = unsigned long baseAddress;

MEMBERDATA = void *baseAdd; // local (as opposed to VME bus) base
address

INCLUDE = <vxWorks.h>;
INCLUDE = <drv/vme/vmechip2.h>;
INCLUDE = <taskLib.h>;
INCLUDE = <vme.h>;
INCLUDE = <sysLib.h>;
INCLUDE = <math.h>;

HXXCODE = {
    extern "C" char *strdup(const char *);
    extern "C" int logMsg();
    extern "C" STATUS vxMemProbe();
    extern "C" double fmod();

};

// ENDCODE

CONSTRUCTCODE = {

int result; // result OK or bus error of a vxMemProbe

// remap the vme bus address to a local bus address
result = sysBusToLocalAdrs(VME_AM_SUP_SHORT_IO, (char *)baseAddress,
                           (char **)&baseAdd);

if (result != OK)
    logMsg("cogging constructor: sysBusToLocalAdrs fail: %d\n", result);

// Check to see if the board has been installed!
// We want to avoid a nasty bus error if not installed
// Else we indicate success that module is OK with good base Add
char *retVal; // value (not used) returned from vxMemProbe
result = vxMemProbe(baseAdd, VX_READ, sizeof(unsigned short),
&retVal);
if (result != OK) {
    logMsg("cogging constructor: vxMemProbe failed: %d\n", result);
    logMsg("Is VME chassis up? Is base address (0x%x) correct?\n",
baseAdd);
}
else { // It seems to be there...
    // merely indicate success that the module was located in VME space
    logMsg("Synchro DDS: vxMemProbe success\n");
    logMsg("Base address set is : 0x%x\n", baseAdd);
}

_frev = 80000; // start with 80 KHz revolution frequency
_numBunchesRhic = 120;
_bunchXfrRate = 30;

// Now we have initialised the arrays, do persistence of parameter
values.

```

```

// Stores the last settings to Non-Volatile RAM every 5 seconds hence
// continuity of operation is achieved even through reboots
// (saves us reentering vals after a reboot)
ADO::ReadArchive();
ADO::setAllEquipment();

};

// ENDCODE

// ++++++ MEMBERCODE SECTION ++++++
// puts the value that is stored in hardware at the given offset from
// the board offset (the board is accessed by 16bit reads) into val
// OK or error is returned
//
MEMBERCODE int FromHardware(unsigned short offset, unsigned long *val) =
{
    int result;
    unsigned short *fromPtr, *toPtr;

    fromPtr = (unsigned short*)((unsigned long)baseAdd + offset);
    toPtr = (unsigned short*)val;
    result = vxMemProbe(fromPtr+1, VX_READ, 2, toPtr);
    if(result == OK)
        toPtr[1] = fromPtr[0];
    return result;
} // ENDCODE

// copy the data from "val" into the hardware address in offset
// (the board is accessed by 16bit writes)
//

MEMBERCODE int ToHardware(unsigned short offset, unsigned long val) = {
    int result;
    unsigned short *fromPtr, *toPtr;

    toPtr = (unsigned short*)((unsigned long)baseAdd + offset);
    fromPtr = (unsigned short*)&val;
    result = vxMemProbe(toPtr+1, VX_WRITE, 2, fromPtr);
    if(result == OK)
        toPtr[0] = fromPtr[1];
    return result;
} // ENDCODE

MEMBERCODE void calcNewValues() = {
    double temp, deltaSynchro, deltaKicker, deltaF;
    unsigned long countsSynchro, countsKicker, NSynchro, NKicker;
    unsigned long deltaFreqSynchro, deltaFreqKicker;
    double twoE32 = 4294967296.0;

    NSynchro = unsigned (1024*_frev/_bunchXfrRate);
    NKicker = unsigned (NSynchro/2);
    countsSynchro = (NSynchro * 4); // calculate synchro duration
    countsKicker = NKicker; // calculate kicker duration

    // this section calculates the delta frequency applied to the
    // synchro synthesizer
    temp = (19.0/4.0);
    temp = temp/_numBunchesRhic;
    temp = fmod(temp,1);
}

```

```

temp+= 0.25;

deltaSynchro = fmod(temp,1);
deltaSynchro *= (4096.0/NSynchro);

// this section calculates the delta frequency applied to the
// to the 1/4 Frev kicker trigger synthesizer
deltaKicker = (1024.0)/_numBunchesRhic/NSynchro;

// add the delta frequency to the base frequency
deltaFreqSynchro =unsigned((19 - deltaSynchro)*4194304.0);
deltaFreqKicker = unsigned(((1 - (2 * deltaKicker))*1048576.0) + 1);

// calibrate the kicker delta frequency duration
deltaF = unsigned(2*deltaKicker*1048576);
deltaF *= 0.01859664917;
printf("\ndeltaF = %f",deltaF);

countsKicker = unsigned((1024.0 * _frev)/(4 * _numBunchesRhic *
deltaF));

// send all data down to the synthesizers and counters
ToHardware(0x0, 0x00400000); // send Frev to the fiducial DDS
ToHardware(0x4, 0x00400000); // upper and lower values

ToHardware(0x8, 0x00100000); // Values for the kicker trigger
ToHardware(0xC, deltaFreqKicker);

ToHardware(0x10, (19*0x400000)); // Reference freq to the AGS synchro
ToHardware(0x14, deltaFreqSynchro);

ToHardware(0x18, (360 * 0x400000)); // frequency of the Beam synch
clock
ToHardware(0x1C, (360 * 0x400000));

ToHardware(0x20, 0x0);
ToHardware(0x24, countsKicker);
ToHardware(0x28, countsSynchro);
ToHardware(0x2C, 0x0);

// try and update the values on the pet page

FromHardware(0x0, &_freq11raw);
FromHardware(0x4, &_freq12raw);
FromHardware(0x8, &_freq21raw);
FromHardware(0xC, &_freq22raw);
FromHardware(0x10, &_freq31raw);
FromHardware(0x14, &_freq32raw);
FromHardware(0x18, &_freq41raw);
FromHardware(0x1C, &_freq42raw);
FromHardware(0x20, &_count1);
FromHardware(0x24, &_count2);
FromHardware(0x28, &_count3);
FromHardware(0x2C, &_count4);

freq11raw.value.updateAsync();
freq12raw.value.updateAsync();
freq21raw.value.updateAsync();
freq22raw.value.updateAsync();
freq31raw.value.updateAsync();
freq32raw.value.updateAsync();

```

```

freq41raw.value.updateAsync();
freq42raw.value.updateAsync();

count1.value.updateAsync();
count2.value.updateAsync();
count3.value.updateAsync();
count4.value.updateAsync();

}; //ENDCODE

// ++++++ DESTRUCTCODE SECTION ++++++
// This code will become the destructor code for the concrete ADO
// Do in case we unload without shutting down library properly!
DESTRUCTCODE = {

}; // ENDCODE
// DESTRUCTCODE

// ++++++ PARAMETER SECTION ++++++
PARAMETER numBunchesRhic {
    CATEGORY = CONT_SETTING;
    TYPE = ULongType;
    READWRITE = W;
    DESC = Number of bunches to be accelerated int the RHIC;
    FEATURE = valueIsAlwaysOk;
    SETCODE = {
        calcNewValues();
        return OK;
    }; //ENDCODE
}

PARAMETER bunchXfrRate {
    CATEGORY = CONT_SETTING;
    TYPE = ULongType;
    READWRITE = W;
    DESC = Number of bunches to be accelerated int the RHIC;
    FEATURE = valueIsAlwaysOk;
    SETCODE = {
        calcNewValues();
        return OK;
    }; //ENDCODE
}

PARAMETER freqv {
    CATEGORY = CONT_SETTING;
    TYPE = DoubleType;
    READWRITE = W;
    DESC = Revolution frequency in Hz;
    FEATURE = valueIsAlwaysOk;
    SETCODE = {
        calcNewValues();
        return OK;
    }; // ENDCODE
}

PARAMETER freq11raw {
    CATEGORY = CONT_SETTING;
    TYPE = ULongType;
    READWRITE = W;
    DESC = Raw frequency, synthesizer 1, frequency 1;
}

```

```

FEATURE = valueIsAlwaysOk;
SETCODE = {
    int result = ToHardware(0x0, _freq11raw);
    return result;
}; // ENDCODE
GETCODE = {
    return FromHardware(0x0, &_freq11raw);
}; // ENDCODE
}

PARAMETER freq12raw {
    CATEGORY = CONT_SETTING;
    TYPE = ULongType;
    READWRITE = W;
    DESC = Raw frequency, synthesizer 1, frequency 2;
    FEATURE = valueIsAlwaysOk;
    SETCODE = {
        int result = ToHardware(0x4, _freq12raw);
        return result;
    }; // ENDCODE
    GETCODE = {
        return FromHardware(0x4, &_freq12raw);
    }; // ENDCODE
}

PARAMETER freq21raw {
    CATEGORY = CONT_SETTING;
    TYPE = ULongType;
    READWRITE = W;
    DESC = Raw frequency, synthesizer 2, frequency 1;
    FEATURE = valueIsAlwaysOk;
    SETCODE = {
        int result = ToHardware(0x8, _freq21raw);
        return result;
    }; // ENDCODE
    GETCODE = {
        return FromHardware(0x8, &_freq21raw);
    }; // ENDCODE
}

PARAMETER freq22raw {
    CATEGORY = CONT_SETTING;
    TYPE = ULongType;
    READWRITE = W;
    DESC = Raw frequency, synthesizer 2, frequency 2;
    FEATURE = valueIsAlwaysOk;
    SETCODE = {
        int result = ToHardware(0xC, _freq22raw);
        return result;
    }; // ENDCODE
    GETCODE = {
        return FromHardware(0xC, &_freq22raw);
    }; // ENDCODE
}

PARAMETER freq31raw {
    CATEGORY = CONT_SETTING;
    TYPE = ULongType;
    READWRITE = W;
    DESC = Raw frequency, synthesizer 3, frequency 1;
    FEATURE = valueIsAlwaysOk;
}

```

```

SETCODE = {
    int result = ToHardware(0x10, _freq31raw);
    return result;
}; // ENDCODE
GETCODE = {
    return FromHardware(0x10, &_freq31raw);
}; // ENDCODE
}

PARAMETER freq32raw {
    CATEGORY = CONT_SETTING;
    TYPE = ULongType;
    READWRITE = W;
    DESC = Raw frequency, synthesizer 3, frequency 2;
    FEATURE = valueIsAlwaysOk;
    SETCODE = {
        int result = ToHardware(0x14, _freq32raw);
        return result;
    }; // ENDCODE
    GETCODE = {
        return FromHardware(0x14, &_freq32raw);
    }; // ENDCODE
}

PARAMETER freq41raw {
    CATEGORY = CONT_SETTING;
    TYPE = ULongType;
    READWRITE = W;
    DESC = Raw frequency, synthesizer 4, frequency 1;
    FEATURE = valueIsAlwaysOk;
    SETCODE = {
        int result = ToHardware(0x18, _freq41raw);
        return result;
    }; // ENDCODE
    GETCODE = {
        return FromHardware(0x18, &_freq41raw);
    }; // ENDCODE
}

PARAMETER freq42raw {
    CATEGORY = CONT_SETTING;
    TYPE = ULongType;
    READWRITE = W;
    DESC = Raw frequency, synthesizer 4, frequency 2;
    FEATURE = valueIsAlwaysOk;
    SETCODE = {
        int result = ToHardware(0x1C, _freq42raw);
        return result;
    }; // ENDCODE
    GETCODE = {
        return FromHardware(0x1C, &_freq42raw);
    }; // ENDCODE
}

PARAMETER count1 {
    CATEGORY = CONT_SETTING;
    TYPE = ULongType;
    READWRITE = W;
    DESC = the first countdown counter;
    FEATURE = valueIsAlwaysOk;
    SETCODE = {

```

```

        return ToHardware(0x20, _count1);
    } // ENDCODE
    GETCODE = {
        return FromHardware(0x20, &_count1);
    } // ENDCODE
}

PARAMETER count2 {
    CATEGORY = CONT_SETTING;
    TYPE = ULongType;
    READWRITE = W;
    DESC = the second countdown counter;
    FEATURE = valueIsAlwaysOk;
    SETCODE = {
        return ToHardware(0x24, _count2);
    } // ENDCODE
    GETCODE = {
        return FromHardware(0x24, &_count2);
    } // ENDCODE
}

PARAMETER count3 {
    CATEGORY = CONT_SETTING;
    TYPE = ULongType;
    READWRITE = W;
    DESC = the third countdown counter;
    FEATURE = valueIsAlwaysOk;
    SETCODE = {
        return ToHardware(0x28, _count3);
    } // ENDCODE
    GETCODE = {
        return FromHardware(0x28, &_count3);
    } // ENDCODE
}

PARAMETER count4 {
    CATEGORY = CONT_SETTING;
    TYPE = ULongType;
    READWRITE = W;
    DESC = the fourth countdown counter;
    FEATURE = valueIsAlwaysOk;
    SETCODE = {
        return ToHardware(0x2C, _count4);
    } // ENDCODE
    GETCODE = {
        return FromHardware(0x2C, &_count4);
    } // ENDCODE
}

ENDDEF;

```